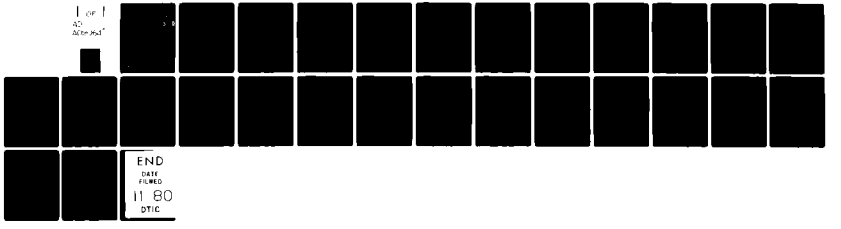
AD-A090 641 BOEING AEROSPACE CO SEATTLE WA ENGINEERING TECHNOLOGY DIV F/G 16/4
NEW TECHNIQUES FOR TEST DEVELOPMENT FOR TACTICAL AUTOPILOTS USI--ETC(U)
JUL 80 E H SHEMETA

UNCLASSIFIED NL



END
DATE FILMED
11 80
DTIC

NEW TECHNIQUES FOR TEST DEVELOPMENT FOR TACTICAL AUTOPILOTS USING MICROPROCESSORS

Edward H. Shemeta
Engineering Technology
Boeing Aerospace Company
Seattle, Washington 98124

SUMMARY

Advances in semiconductor technology have made it possible to introduce increasingly complex digital logic in avionics, weapons and control systems. However, the testing of such systems represents a new problem. Test requirements of missile systems are now approaching the complexity of aircraft themselves a decade ago. The costs of creating test procedures for the new designs will be prohibitive if new methods of test generation are not developed.

In the early 1970's, simulation methods were developed to generate test patterns on computers to increase productivity. These methods, e.g., 'LASAR','FAIRTEST', etc. were used successfully to support the A7-D/E and the S3-A aircraft test development needs. These methods have been also used to develop tests for E3-A, B-1 Avionics and IUS programs at Boeing.

As useful as these methods are, the computer costs for test generation are becoming high. Also, the time required to create computer models needed to support the test generation process itself has become very long. It has been estimated that to develop gate-level models for off the shelf LSI (Large Scale Integration) microprocessors and similar devices would take a man-year. Since the development phases of such programs are a year or less, the methods which worked so well in the past are not keeping up with the new requirements.

At Boeing, research in the test development area has been directed at developing new approaches that are applicable to the new technology. The most promising approach is based on the use of algorithmic test generation techniques instead of the classic truth table methods.

Using algorithms as the source for test code, it is not necessary to store millions of test vectors in a tester or to run costly simulations to generate the test vectors.

This paper reports on a demonstration of the application of the method to generate system level tests for a typical tactical missile autopilot. The test algorithms are based on the autopilot control law. When loaded on the tester with appropriate control information, the

complete autopilot is tested to establish if the specified control law requirements are met. Thus, the test procedure not only checks to see if the hardware is functional, but also checks the operational software. The technique also uses a 'learning' mode to allow minor timing or functional deviations from the expected responses to be incorporated in the test procedures.

A potential application of this test development technique is the extraction of production test data for the various subassemblies. The technique will 'learn' the input-output patterns forming the basis for development and production tests. If successful these new techniques should allow the test development process to keep pace with semiconductor progress.

INTRODUCTION

Technology advances in semiconductor design and manufacturing techniques have made the miniaturization of high peformance systems a practical and economical concept. The replacement of marginal and sensitive analog hardware, such as servomechanisms and potentiometers in computing circuits, with digital processing circuits has impacted all aspects of electronic products. The result is the emergence of a whole new generation of 'smart' instruments. The commercial success of electronic games, etc., is a testimonial to the cost-effectiveness and versatility of the new digital techniques.

The application of this technology to low cost digital autopilots for small, highly accurate, terminally-guided missiles is particularly advantageous. A reliable low cost 'smart' missile which is field adaptable to a broad range of threats could be the ultimate next generation field weapon.

However it is not enough for a system to be constructed from low cost parts and be functionally versatile. The system must be manufacturable, to assure low cost, and, more importantly, field supportable to achieve its potential as a weapon.

Unfortunately, systems based on microprocessor technology present new problems in developing effective production test procedures using traditional methods. The problem lies in the very advantages of such systems, ie., the ability to change the system programming to meet new requirements without changing the hardware. As the software changes are made, the test procedures must likewise be modified. It is imperative that the system test procedures be capable of being modified as rapidly and effectively as the microprocessor software.

In this paper the traditional test approaches are reviewed as to their adequacy in meeting the new requirements, and a new approach, algorithmic test generation, will be outlined. This approach shows promise in answering these new requirements. An example of algorithmic test generation will be developed to illustrate the development of test algorithms from system level design data.

STATE OF THE ART IN DIGITAL TEST DEVELOPMENT

Early test generation methods in the vacuum tube era were based on signal tracing techniques using scopes and waveform diagrams. Standard input signals were generated and data flow was manually traced from input to output. With good access to the vacuum tube sockets, the test procedures were easy to develop and use.

With the advent of integrated circuits, the simple signal tracing concepts were more difficult to document and the inability to trace the signals internal to the chip made the mechanics of signal tracing peculiar to the individual design approaches. This meant that the maintenance personnel had to be trained to diagnose each peculiar system.

At that time the concept of developing tests for digital logic to detect logic faults was developed. [1] The stuck-at-one, stuck-at-zero fault concept was introduced and computer simulation methods were used to evaluate test sequence effectiveness.

Following the development of simulators, various methods for automatic test generation were developed. One of the first applications of such techniques was the development of tests for the A7-D/E aircraft avionics and its peculiar ground support equipment (PGSE).

For example, in this application [2] the logic schematics were encoded (Figure 1) as the source data for the logic simulation and test generation process. Figures 2 and 3 are a listing of the input code needed to generate tests for the unit. Figure 4 is the test sequence generated for the schematic and Figure 5 is the report of the test effectiveness for that test sequence. The test patterns are then converted to the ATE (automatic test equipment) source code and are ready for use on the ATE.

Such methods have been applied very successfully in the last decade on a number of aircraft programs. Boeing programs have made extensive use of various commercial test generation techniques as have virtually all major avionics manufacturers.[3] The military depots have, in like manner, used these techniques and have reported order of magnitude cost reductions over manual techniques. [4]

Since, the advent of LSI and VLSI (very large scale integration) technology and the application of microprocessors in systems, it has become increasingly difficult to develop the necessary simulation models needed to generate tests. The computer and model development costs can exceed the value of the equipment itself.

Because of the dynamic state of development in the semiconductor industry, models developed for simulation and test generation can become obsolete before the models can be developed and validated. In the past, once a "74/54" series part-library was developed, few changes were needed and the modeling costs could be written off against many users. In the current VLSI state of affairs, for custom designs, models

will probably be used only once, resulting in high set-up costs for test generation as well as expensive computer runs due to the large number of gates needed in the models.

Aside from the cost and modeling difficulties for the new microprocessor based designs, there is some question of the adequacy of the tests developed using the stuck fault concept.

Most test generation techniques are designed to develop static tests only. These simulators only approximate the time domain behavior of logic gates. Since 30 to 40 percent of the failure modes in VLSI are dynamic in nature, a test generation program reporting 100% of the "stuck-at" faults detected is probably only 70% effective for the total fault universe. This means that the remaining untested faults must be detected and isolated at the next higher assembly level. The usual symptom of such a problem is that the UUT (unit under test) will work in one sytem but not another. But when returned for test on the ATE, it always passes the test.

There are two other considerations for microprocessor systems that are unique. Such a system can be tested to exhaustion at the part level and subassembly level but yet fail in the system. Because microprocessor systems consist of hardware and firmware, testing the hardware and firmware independently does not mean that when the two are integrated the system will function as expected. Microprocessors exhibit failure modes, called pattern sensitivity, which are dependent upon the instruction sequence executed. Thus if two programmers were to write code for the same application, it is possible that one program could function normally and the other fail due to the peculiar sequence of instructions used by one programmer's code and not by the other.

For these reasons the built-in test concept, where the microprocessor exercises itself with a stored program, can only check out a fraction of the total failure modes.

The ideal approach is one where a complete checkout of the hardware and firmware, within the context of a dynamic test, can be performed. Further, the test generation technique should be capable of rapid modification as the system design matures, and be inexpensive to implement. Hopefully it would be a natural extention of the normal system synthesis and engineering process. The algorithmic test generation procedure is an approach which can meet these goals for a large class of systems.

ALGORITHMIC PATTERN GENERATION

Algorithmic pattern generators have found wide spread use in memory testing. Because of the highly ordered structures of memories, the efficiency of testing is greatly improved by using special hardware pattern generators instead of the classic truth tables. With the introduction of programmable pattern generators, millions of tests can be generated from a few input parameters. This also obliviates the need to store the truth tables in expensive ATE.

As an example, to keep pace with memory development, a number of algorithms have been developed, ie., "ping-pong", "butterfly", etc. Such techniques have reduced the cost of test equipment and made it possible to effectively test the most complex memories with a minimum of set-up costs. However, because of the differences in the design and manufacturing methods of memories, there is still no single completely accepted sequence of test algorithms available to completely test a particular semiconductor memory part.

The algorithms needed to perform a specific type of test for memories are well-known. For the type of systems under consideration, ie., real-time microprocessor based control systems, no general pattern generation techniques have been developed to date. Most of the test procedures are based on exercising the instruction set of the microprocessor, or a simple input/output wrap-around test.

For the class of applications that are describable by transfer functions, it is possible to generate a set of algorithms that can create test patterns in a manner similar to that described for memory tests. Further, these patterns are dynamic in nature, and can be used to checkout the hardware and firmware to certify it's integrated adequactly for a particular application.

The advantage of algorithmic approach is that faults in the system which have no effect on the intended use do not cause the unit to fail the test. For example, if a particular instruction of a microprocessor was faulty but it is not used, the fault is a 'don't care' fault. (If two programs were written for the same application and one programmer used the faulty instruction and the other did not and there were no coding errors, one would pass the test, the other would fail.) In like manner, if a memory cell were faulty and it was never used, the fault would be a 'don't care'. If the "stuck-at" test concept were used to develop tests, a useable unit could be rejected even though it could adequately perform the intended mission. Such rejections tend to increase the overall cost of systems.

The algorithmic test pattern generation concept tests the suitability of the total system for an application, including real time software and hardware performance. However, since the test procedure is application dependent, it does not assure the hardware is adequate for other uses. An advantage of the technique is that it is not difficult to synthesize the algorithms needed to create the desired test patterns for subsequent users.

ALGORITHM SYNTHESIS

Systems are frequently described in block diagram form with transfer functions as shown in Figure 6. Generally the system under test (SUT) and the "plant" or the controlled object are easily partitioned into independent and distinct transfer functions. For example, if the controller is digital, the transfer functions are usually represented in 'Z- Transform' form and the 'plant' in conventional LaPlace transforms. Generally, the systems analysis activities have developed numerical simulation equations for the system. But these are usually too complex or more detailed than needed for this application.

The first step of the process is to convert the SUT transfer functions into difference equations. These equations will be used to develop the outputs of the SUT, given the inputs. Since the Z-Transform parameters contain the sampling rate, the test rate is constrained to be that used in generating the Z-Transforms.

If the SUT transfer functions are LaPlace transforms, they must be converted to Z-Transforms. Generally, the step-invariant transform will provide an exact time-domain solution for most situations. The procedure for performing these mathematical operations are well-known. [5,6]

The difference equations developed at this step must be exact, because they form the basis for the algorithms that will generate test patterns for the SUT. The algorithms will probably generate test patterns more precise than those implemented in the microprocessor program itself, so the test procedure provides a quality control measure for the numerical methods used in the software. Such a quantitative measure is a valuable by-product of this testing approach.

Since the SUT generally has dynamic feedback signals from the plant being controlled, its transfer functions must likewise be translated into difference equations. These equations will generate the necessary feedback signals needed to dynamically test the SUT in the feedback loop.

However, these equations or models need not be as precise as those for the SUT, but need only be a reasonable approximation of the environment the SUT drives. Thus, in many situations the classic underdamped quadratic transfer function can be replaced by a single low-pass pole. Since stability analysis is not the goal, but rather the generation of feedback signals, such approximations greatly simplify the synthesis of the test generation algorithms without reducing the validity of the test procedure.

AN EXAMPLE

Figure 7 is a block diagram of an autopilot of an experimental missile, the T6. The T6 airframe is a 70 pound modular test vehicle designed to accept a variety of six inch diameter seekers. The T6 autopilot is typical of the type of real-time control system amenable to algorithmic test generation techniques.

For purposes of illustration the pitch channel will be used in the example, but the method can readily be extended to include all the channels and modes of operation. Figure 8 is a block diagram of the pitch channel of the autopilot. The first block is the pitch contoller, the second block the actuator controller. Both of these functions are contained in the microprocessor software and are the objects of the test procedure. The other blocks are the missile fin actuator and the missile aerodynamics. The feedback to the autopilot is via a gyro which is assumed to have the transfer function of an integrator.

An algorithm for the first block is developed in equation (1) by expanding the Z-transform into a difference equation in the usual manner. The limiters are coded as shown in equation (2) and in like manner the actuator is expressed as shown in equation (3).

```
R(I)=[C(I)-S(I)]+(1-.9182)*[C(I-1)-S(I-1)]-0.9182*[C(I-2)-
 S(I-2)]-(.6434-.2302)*R(I-1)+(.2302)*( 6434)*R(I-2)   (1).

IF R(I)>16/.18 THEN R(I)=16/.18                        (2).

IF R(I)<-16/.18 THEN R(I)=-16/.18                      (3).

A(I)=R(I)-F(I)-.667*[R(I-1)-F(I-1)]                    (4).

A(I)=.2184*A(I)                                        (5).

IF A(I)>22.5/.18 THEN A(I)=22.5/.18                    (6).

IF A(I)<-22.5/.18 THEN A(I)=-22.5/.18                  (7).

F(I)=.6*A(I)-.4*F(I)                                   (8).

C(I)=K*E*F(I)+2*F*C(I-1)-G*C(I-2)                      (9).
```

Since the objective of the test is to checkout the autopilot hardware and software, a single pole low pass filter is used to represent the actuator in equation (8). A quadratic transfer function in equation (9) is derived from the Z-transform of the LaPlace transfer function. An integrator is used to approximate the missile aerodynamics.

The scale factors are important (0.18 deg./LSB in the example) since these values are used to define the value of the least significant bit of the data generated by the tester. With the gains fixed as shown for the test procedure, the scale factors for the inputs and outputs are easily calculated

To checkout the algorithms they are converted to a BASIC program and run on a computer. A listing of the program is provided in Figure 9, and the response to a step input is shown in Figure 10.

Once checked out using the BASIC program, the equations need to be modified slightly to be in the source language of the target tester. For this example the test system is Boeing's Sentry VIII test system (Figure 11). The source language is FACTOR, an ALGOL-type language. The conversion to FACTOR is a simple procedure.

To develop the detailed test program the microprocessor hardware interface must be defined. In this example an 8 bit data bus, one bit hold, and ready control are assumed. It should be understood that the detailed design of the particular SUT establishes the definitions. All inputs and outputs are multiplexed on the system data bus.

The first step is the definition of the input/output pins of the SUT. Also control pins are established along with the input output sequencer signals and the data bus control lines. The test rate is set up by tester commands in the manner peculiar to the individual tester The tester driver is on when the "DA" command is active, off when the "DB" command is active. The tester is checking the output data when the "MA" is set and ignoring data when the "MB" is set. In the FACTOR code "DUT[DADB]=1" means the "DB" is set. "DUT[DADB]=0" means the "DA" is set. In like manner, "DUT[MAMB]=0" means the "MA" is set and "DUT[MAMB]=1" means "MB" is set. Other testers will use similar techniques, but it it important that these functions be performed in a dynamic manner (on-the-fly) otherwise the test data will not be vaild.

The FACTOR source code is shown in Figure 11, and a compiled listing in Figure 12. The generated test code is shown in Figure 13. It should be noted that the actual code is significantly longer and only a small portion of the actual code is printed.

GENERALIZATIONS & CONCLUSIONS

The algorithms developed in the example were selected to illustrate the techniques of "algorithmic pattern generation" and only illustrate one of many possible approaches to modeling systems for test generation purposes. The recursive difference equation, based on the Z-Transform calculus, has many advantages in that the theory is highly developed, and is a very natural way to develop a dynamic test.

Those versed in simulation art will find many ways to construct algorithms that do not require transfer functions or the linear system assumptions.

The nature of the algorithms is constrained to a degree by the target tester. The system used in the example, the Sentry VIII manufactured by Fairchild, is particularly easy to use to create test algorithms. The utility program "LMLRN" handles many of the tester control functions in a very simple manner. Other testers may require an additional processing step to perform the test control generation function.

Although not mentioned in the example, it is feasible to expand the input/output definitions (pin lists) and 'learn' the test data at the subassembly levels to form the basis for developing production and field support tests. The tester control functions for such an application would be significantly more complex, but with careful design of the subsystem interfaces, the process can be made very productive.

The approach outlined is compatible with the normal systems engineering approach. Simulation models used to support systems analysis activities are directly applicable to the test generation process. However, further research and analysis is needed to define the adequacy of test procedures developed using functional approaches instead of the classic "stuck-at" methods. The acceptance of equipment with possible manufacturing defects, even though the defects do not impact the intended function, presents certain cultural problems to a quality assurance operation, ie. "Zero-Defects".

REFERENCES

(1) Seshu,S. and Freeman,D.N., "The Diagnosis of Asynchronous Sequential Switching Systems", IRE Trans. Electronic Computers, vol. EC-11, pp. 459-465, August 1962.

(2) TO 51T1-15-8-29, "Program number 47264"

(3) "SELECTION GUIDE FOR DIGITAL TEST PROGRAM GENERATION SYSTEMS", Navmatinst 3960.9A, 27 MARCH 1979.

(4) Pedersen, G. "Reducing ATE Software Development Costs", AUTOTESTCON, Sept. 1979, Minneapolis, Minn.

(5) Breding, D.R. "Digital Simulation of a Seismic System", Proceedings of the 26'th International Instrumentation Symposium, Seattle, Wa., Advances in Test Measurement, Vol 17, pp. 297-299.

(6) Stearns, S.D. 1975, "Exact Digital Models of Continuous Linear Systems", SAND75-8012.

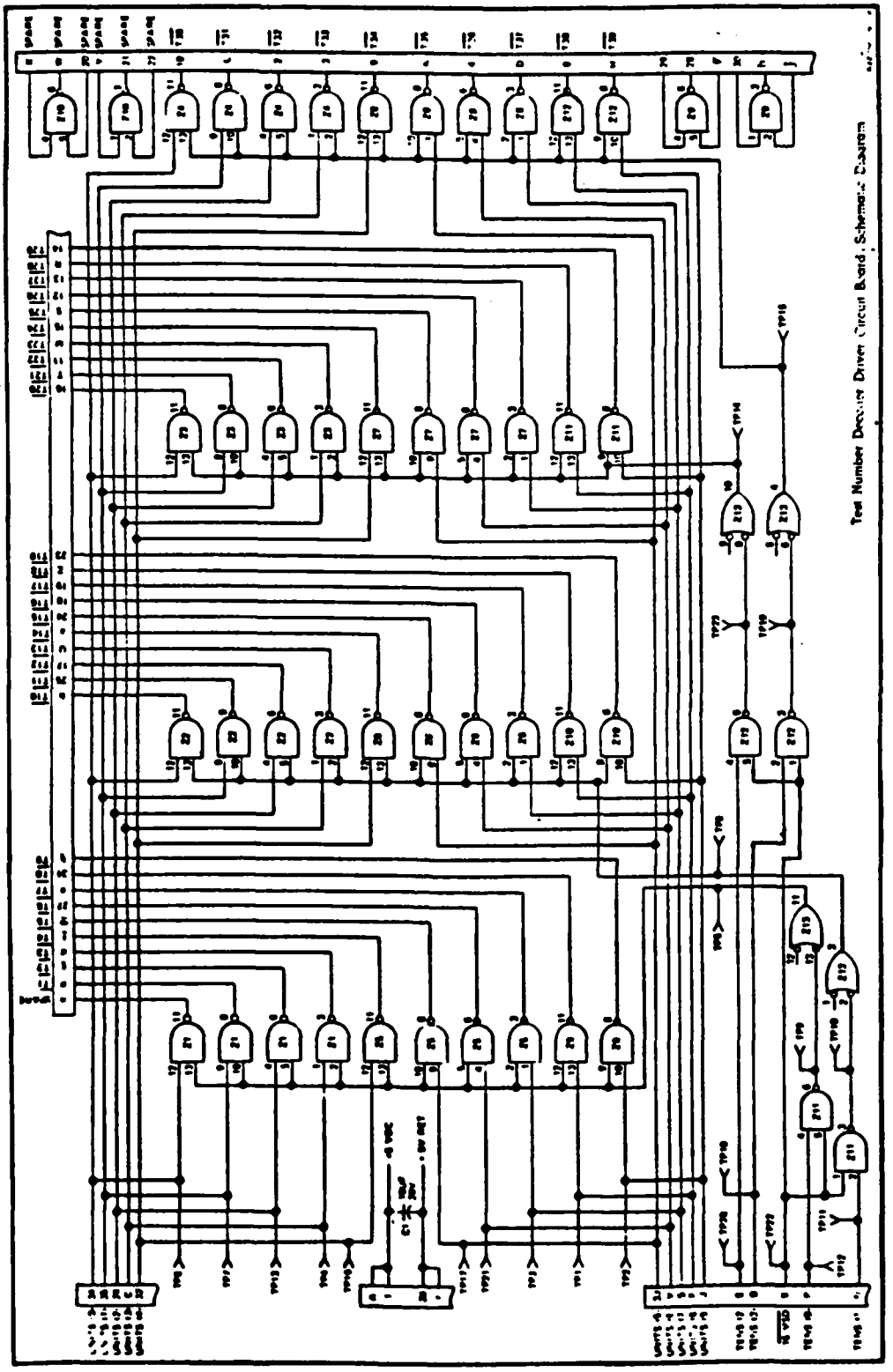


Figure 1. Test Number Decoder/Driver Circuit Board Schematic Diagram—Drawing Number 275280

```
      47264  DWG # 275270    87-49-9002-2040                                          72/223  PAGE   2

FLGS   LOCTR    OBJCODE   ADDR   STMT      SOURCE STATEMENT

                                  4             NETSTART
                                  6   61        DS46   (P#C,Z13S11),(P#26,Z13S11),(P#35,Z13S11),(P#34,Z13S11)
                                 15   62        DS46   (P#C,Z1383),(P#26,Z1383),(P#35,Z1383),(P#34,Z1383)
                                 24   63        DS46   (P#C,Z1381C),(P#26,Z13810),(P#35,Z1381C),(P#34,Z13810)
                                 33   64        DS46   (P#C,Z1384),(P#26,Z1384),(P#35,Z1384),(P#34,Z1384)
                                 42   65        DS46   (P#S,Z13S11),(P#V,Z13S11),(P#33,Z13S11),(P#32,Z13S11)
                                 51   66        DS46   (P#S,Z1383),(P#V,Z1383),(P#33,Z1383),(P#32,Z1383)
                                 60   67        DS46   (P#S,Z13810),(P#V,Z1381C),(P#33,Z1381C),(P#32,Z13810)
                                 69   68        DS46   (P#S,Z1384),(P#V,Z1384),(P#33,Z1384),(P#32,Z1384)
                                 78   69        DS46   (P#3C,P#JJ),(P#29,P#FF),(Z13811,P#JJ),(Z13811,P#F)
                                 87   610       DS46   (P#Y,P#22),(P#X,P#27),(Z1383,P#JJ),(Z1383,P#F)
                                 96   611       DS46   (P#A,P#N),(P#P,P#Q),(Z13810,P#JJ),(Z1381C,P#F)
                                105   612       DS46   (P#E,P#D),(P#E,P#G),(Z1384,P#JJ),(Z1384,P#F)
                                114   613       DS37   (,Z11831,(,Z12831,(Z12861,(,Z11861
                                122   P#NN      EQ     Z1S11
                                124   P#PP      EQ     Z198
                                126   P#LL      EQ     Z116
                                128   P#JU      EQ     Z183
                                130   P#LL      EQ     Z5S11
                                132   P#MM      EQ     Z918
                                134   P#27      EQ     Z31E
                                136   P#EE      EQ     Z913
                                138   P#31      EQ     Z5S11
                                140   P#KK      EQ     Z918
                                142   P#UU      EQ     Z2S11
                                144   P#25      EQ     Z218
                                146   P#J       EQ     Z213
                                148   P#17      EQ     Z21E
                                150   P#AA      EQ     Z6S11
                                152   P#24      EQ     Z618
                                154   P#18      EQ     Z616
                                156   P#19      EQ     Z613
                                158   P#Z       EQ     Z1S11
                                160   P#23      EQ     Z1C18
                                162   P#16      EQ     Z3S11
                                164   P#T       EQ     Z31E
                                166   P#11      EQ     Z31E
                                168   P#M       EQ     Z313
                                170   P#15      EQ     Z7S11
                                172   P#S       EQ     Z718
                                174   P#12      EQ     Z716
                                176   P#13      EQ     Z713
                                178   P#H       EQ     Z11S11
                                180   P#14      EQ     Z1118
                                182   P#W       EQ     Z1116
                                184   P#21      EQ     Z1113
                                186   P#1P      EQ     Z4S11
                                188   P#L       EQ     Z418
                                190   P#2       EQ     Z41E
                                192   P#3       EQ     Z413
                                194   P#Y       EQ     Z8S11
                                196   P#K       EQ     Z818
                                198   P#4       EQ     Z81E
                                200   P#U       EQ     Z813
                                202   P#D       EQ     Z12S11
```

P#U

PAD

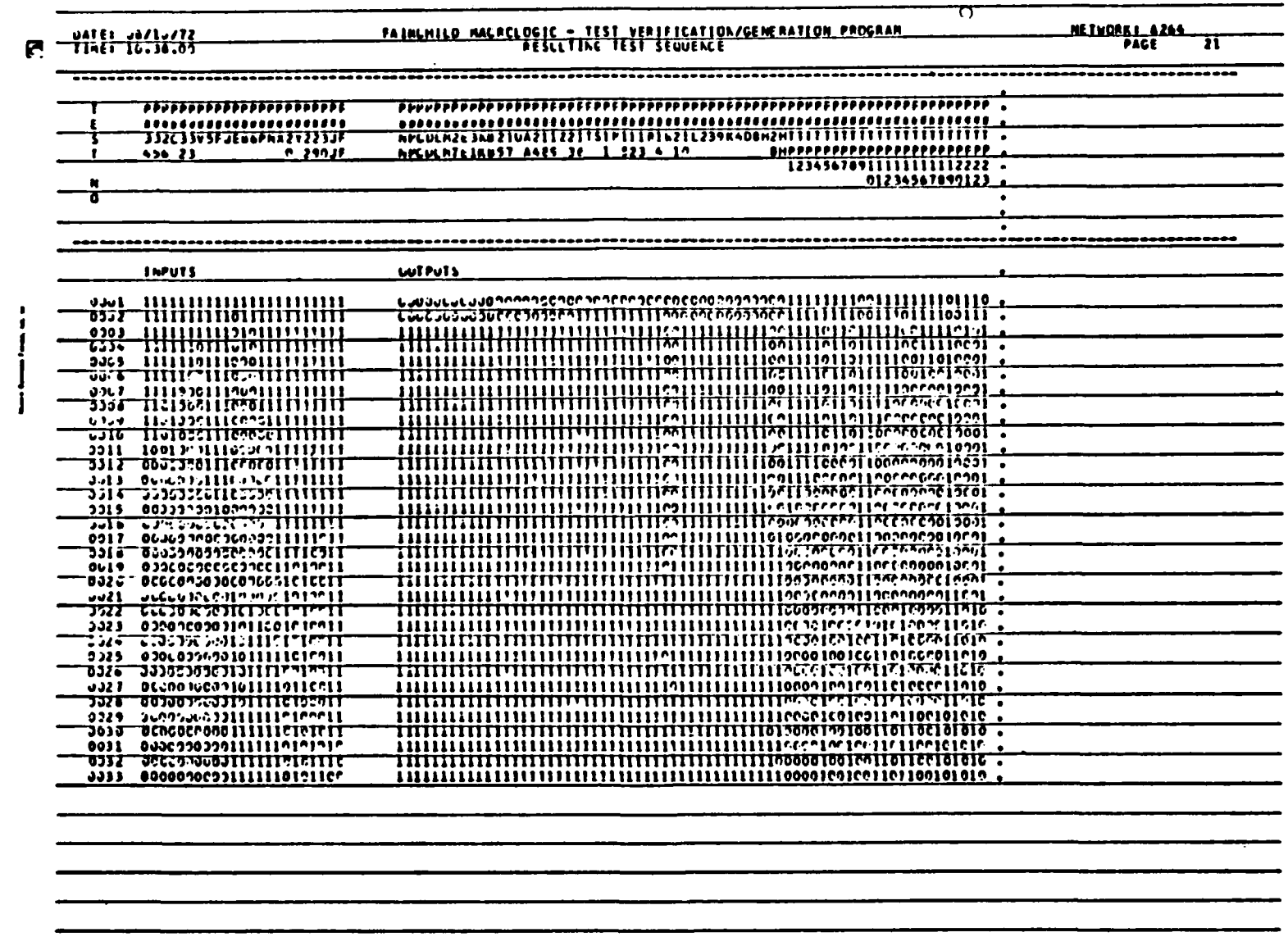
Figure 2. Network

```
      01204  DWG 0 273270    07-49-9002-2040                                      72/223  PAGE    3

PLGS  L.CTR   OBJCODE   ADDR  STMT        SOURCE STATEMENT

                              204  P0H        EQ      Z1210
                              206  P026       EQ      Z516
                              208  P0MM       EQ      Z463
                              210  P0TP1      EQ      P0F
                              212  P0TP2      EQ      P0J
                              214  P0TP3      EQ      P09
                              216  P0TP4      EQ      P0C
                              218  P0TP5      EQ      Z13011
                              220  P0TP6      EQ      P034
                              222  P0TP7      EQ      P035
                              224  P0TP8      EQ      Z1303
                              226  P0TP9      EQ      Z1110
                              228  P0TP10     EQ      Z1103
                              230  P0TP11     EQ      P0A
                              232  P0TP12     EQ      P0P
                              234  P0TP13     EQ      P026
                              236  P0TP14     EQ      Z1301C
                              238  P0TP15     EQ      Z1304
                              240  P0TP16     EQ      P032
                              242  P0TP17     EQ      P033
                              244  P0TP18     EQ      P00
                              246  P0TP19     EQ      Z1203
                              248  P0TP20     EQ      P0E
                              250  P0TP21     EQ      P0V
                              252  P0TP22     EQ      P0E
                              254  P0TP23     EQ      Z1210
                              256  P0A        VCC
                              257  P036       GND
                              258  P0MM       GND
                              259             VERIN  P034,P035,P026,P0C,P032,P033,P0V,P05,P0F,P0J,P0E,P0B,    *
                              260                    P04,P0P,F0N,P0X,P020,P0Y,P022,P029,P030,P0JJ,P0FF
                              307             VEROUT P0NN,P0PP,P0CC,P0DD,P0LL,P0MM,P027,P0EE,P031,P0KK,P0BB,*
                              308                    P025,P017,P0U,P0AA,P024,P010,P019,P0Z,P023,P016,P0T,P05,*
                              309                    P011,P0P,P015,P012,P013,P0R,P014,P0W,P021,P017,P0L,P02,  *
                              310                    P03,P09,P0K,P04,P0U,P08,P0H,P026,P0MM,P0TP1,P0TP2,P0TP3,*
                              311                    P0TP4,P0TP5,P0TP6,P0TP7,P0TP8,P0TP9,P0TP10,P0TP11,        *
                              312                    P0TP12,P0TP13,P0TP14,P0TP15,P0TP16,P0TP17,P0TP18,P0TP19,*
                              313                    P0TP20,P0TP21,P0TP22,P0TP23
                              448             RETEND
                              467                    379 SIGNALS ASSEMBLED = 03.79 PCT OF FAIRSIM CAPACITY
                              469             END
```

Figure 3. Network (Concluded)



Page 13

Figure 4. Resulting Test Sequence

DATE: 08/10/72 PAIMEMILO MACROLOGIC - TEST VERIFICATION/GENERATION PROGRAM NETWORK A2A5
TIME: 15.38.40 RESULT SUMMARY PAGE 33

INDIVIDUAL DEFECTS

TYPE	INCLUDING S-SIGNALS TOTAL NUMBER	NUMBER CONSIDERED	NUMBER DETECTED	PERCENT DETECTED	UNDEFINED OUTPUT	EXCLUDING S-SIGNALS TOTAL NUMBER	NUMBER CONSIDERED	NUMBER DETECTED	PERCENT DETECTED	UNDEFINED OUTPUT
STUCK OUTPUTS	238	238	238	100.00	0	238	238	238	100.00	0
STUCK INPUTS	334	334	334	100.00	0	334	334	334	100.00	0
SHORTS	0	0	0	****	0	0	0	0	****	0
OPENS	0	0	0	****	0	0	0	0	****	0
TOTAL	572	572	572	100.00	0	572	572	572	100.00	0

EQUIVALENCE CLASSES

TYPE	INCLUDING S-SIGNALS TOTAL NUMBER	NUMBER CONSIDERED	NUMBER DETECTED	PERCENT DETECTED	UNDEFINED OUTPUT	EXCLUDING S-SIGNALS TOTAL NUMBER	NUMBER CONSIDERED	NUMBER DETECTED	PERCENT DETECTED	UNDEFINED OUTPUT
CLASSES INCLUDING AT LEAST ONE STUCK OUTPUT DEFECT	150	150	150	100.00	0	150	150	150	100.00	0
CLASSES INCLUDING AT LEAST ONE STUCK INPUT DEFECT	242	242	242	100.00	0	242	242	242	100.00	0
CLASSES INCLUDING AT LEAST ONE SHORT DEFECT	0	0	0	****	0	0	0	0	****	0
CLASSES INCLUDING AT LEAST ONE OPEN DEFECT	0	0	0	****	0	0	0	0	****	0
ALL EQUIVALENCE CLASSES	246	246	246	100.00	0	246	246	246	100.00	0

Figure 5. Test Program Scope

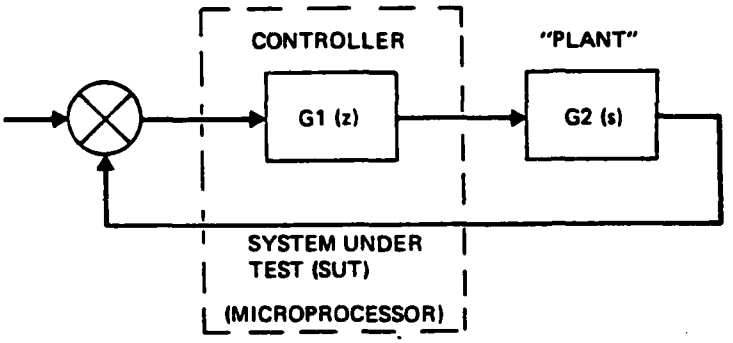


Figure 6. Block Diagram—Typical System

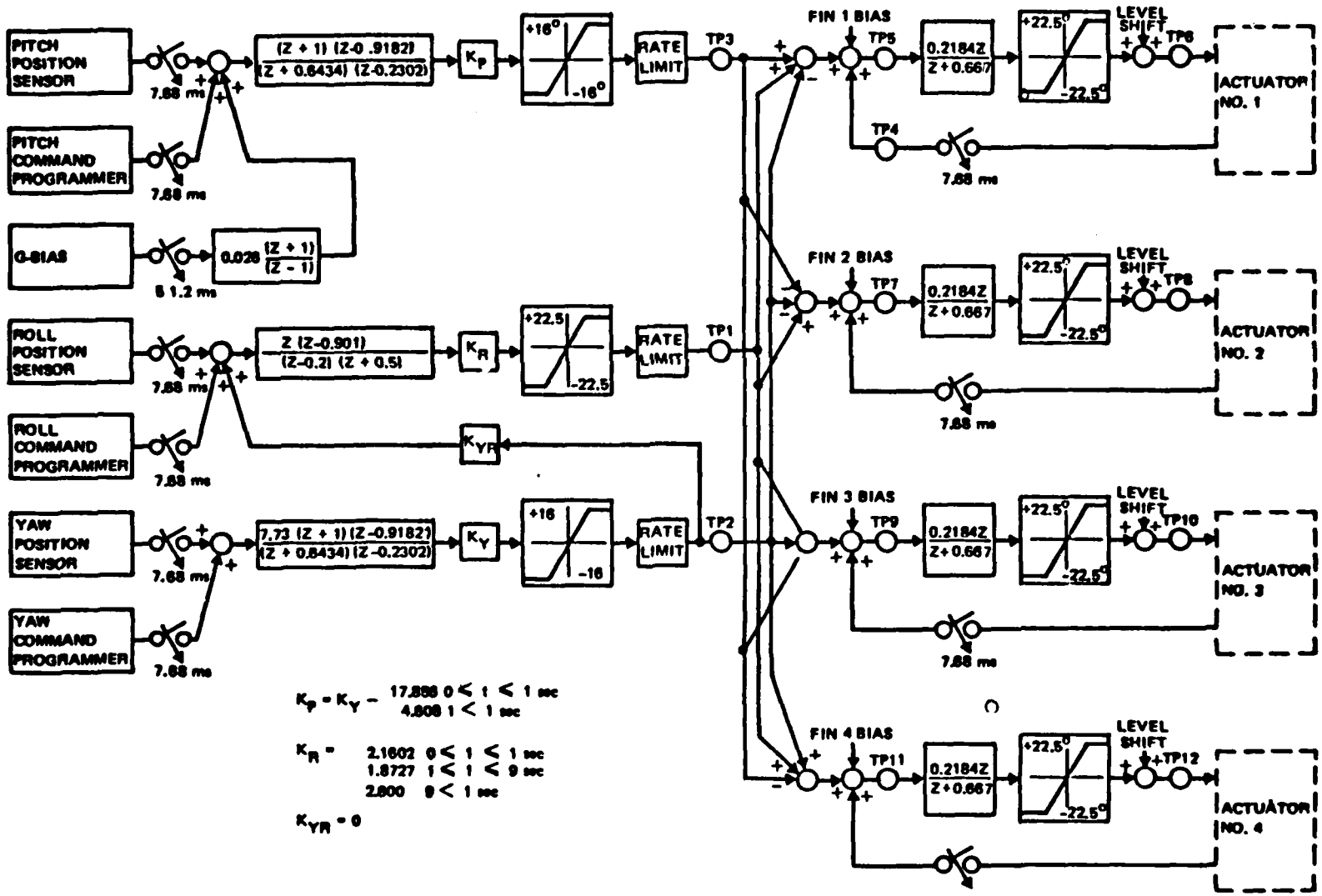


Figure 7. Autopilot Functional Diagram

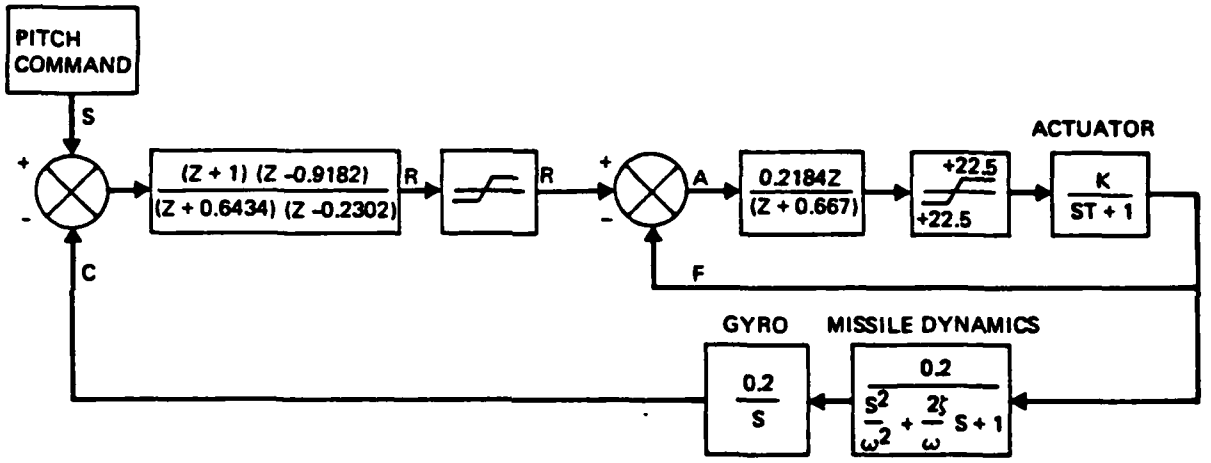


Figure 8. Block Diagram of Pitch Channel

FIGURE 9 "BASIC" SOURCE CODE FOR TEST ALGORITHM CHECKOUT

```
1 REM ********************************************************
2 REM    DEFINE SIMULATION VARIABLES
3 REM
4 REM    R=AUTOPILOT CONTROLLER OUTPUT
5 REM    C=MISSILE FEEDBACK
6 REM    A=ACTUATOR CONTROL SIGNAL
7 REM    F=FIN FEEDBACK SIGNAL
8 REM    S=COMMAND INPUT
9 REM ********************************************************
10 DIM R(3),C(3),A(3),F(3),S(3),X(3)
20 REM ********************************************************
30 REM   INITIALIZE ALL VARIABLES
40 REM ********************************************************
50 MAT R=ZER
60 MAT C=ZER
70 MAT A=ZER
80 MAT F=ZER
90 MAT S=ZER
95 MAT X=ZER
100 REM ********************************************************
110 REM SIMULATION OF CONTROLLER USING DIFFERENCE EQUATIONS
120 REM ********************************************************
150 REM
160 I=3
200 FOR J=1 TO 100
205 IF J=3 THEN S(3)=10
210 R(I)=C(I)+.0818*C(I-1)-.9182*C(I-2)-.4132*R(I-1)+.1481*R(I-2)
250 REM
260 REM ********************************************************
270 REM     GAIN AND LIMIT CALCULATIONS FOR CONTROLLER
280 REM ********************************************************
290
320 IF R(I)>16/.18 THEN R(I)=16/.18
330 IF R(I)<-16/.18 THEN R(I)=-16/.18
340 REM
350 REM ********************************************************
360 REM  ACTUATOR DIFFERENCE EQUATIONS
370 REM ********************************************************
380 REM
400 A(I)=R(I)-F(I)-.667*(R(I-1)-F(I-1))
410 A(I)=.2184*A(I)
405 REM ********************************************************
406 REM  ACTUATOR GAIN AND LIMIT CALCULATIONS
407 REM ********************************************************
420 IF A(I)>22.5/.18 THEN A(I)=22.5/.18
430 IF A(I)<-22.5/.18 THEN A(I)=-22.5/.18
440 REM
450 REM ********************************************************
460 REM  FIN SERVOCALCULATIONS
470 REM ********************************************************
480
```

```
480 REM
500 F(I)=.6*A(I)-.4*F(I)
510 REM
520 REM ******************************************************
530 REM MISSILE FLIGHT CALCULATIONS
531 REM
532 REM MISSILE NATURAL FREQ.=120 RADIANS/SEC.
533 REM
534 REM MISSILE DAMPING FACTOR =.7
535 REM
536 REM GAIN CONSTANT = .2
540 REM ******************************************************
550 REM
560 K=1.24882
570 E=.313568
580 F=.458459
590 G=.30851
600 C(I)=.2*K*E*F(I)+2*F*C(I-1)-G*C(I-2)
605 X(I)=.2*C(I)+X(I)
606 C(I)=S(I)-X(I)
610 REM
620 REM ******************************************************
630 REM  TUMBLE VARIABLES
640 REM ******************************************************
650 REM
655 PRINT USING 741;J,S(3),X(3),R(3),A(3),F(3),C(3)
660 R(1)=R(2)
670 R(2)=R(3)
680 C(1)=C(2)
690 C(2)=C(3)
700 A(1)=A(2)
710 A(2)=A(3)
720 F(1)=F(2)
730 F(2)=F(3)
741 IMAGE 7(XDDD.DX)
800 NEXT J
9000 END
```

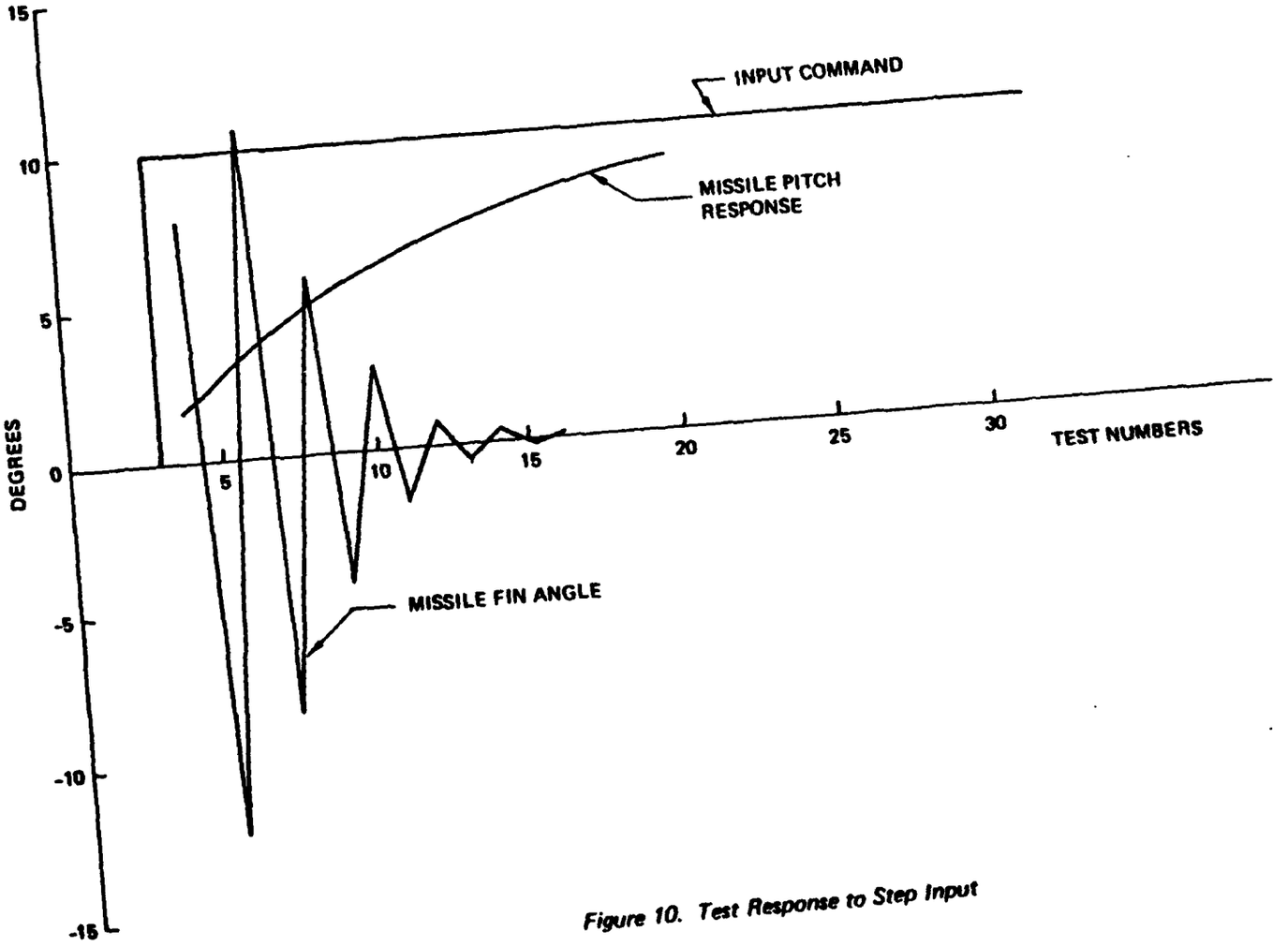


Figure 10. Test Response to Step Input

FIGURE 11 ALGORITHMIC TEST PATTERNS FOR SENTRY VIII TESTER

```
SET PAGE 4095;

REM ************************************************
    * SET UP PIN DEFINITIONS FOR ALGORITHMIC      *
    * TEST PATTERN GENERATION                     *
    ************************************************;

DCL PDATA<8>/10,11,12,13,14,15,16,17/;
DCL PHLD<1>/5/;
DCL PRDY<1>/3/;

DCL DUT<5>;

 DATA=1;
 HLD=2;
 RDY=3;
 DADB=4;
 MAMB=5;

EXEC LMMOD(0);   REM INITIALIZE TESTER MEMORY;

EXEC LMLRN(1,PDATA,PHLD,PRDY,1,2);

REM SET UP PIN DEFINITIONS IN TESTER;

REM************************************************
   *   DEFINE SIMULATION VARIABLES               *
   ************************************************;

DCL R<3>;
DCL C<3>;
DCL A<3>;
DCL F<3>;
DCL S<3>;
DCL X<3>;

REM************************************************
   *  SIMULATION OF CONTROLLER USING DIFF. EQU.  *
   ************************************************;

I=3;
DUT<HLD>=0;          REM START PROCESSOR;
DUT<RDY>=1;
DUT<DADB>=0;
DUT<MAMB>=1;

EXEC LMLRN(4,M,DUT);    REM GENERATE A TEST;
```

```
M=M+1;                    REM TEST NUMBER COUNTER;

FOR J=1 THRU 100 DO BEGIN
IF J EQ 3 THEN S<I>=10;

R<I>=C<I>+.0818*C<I-1>-.9182*C<I-2>-.4132*R<I-1>
     +.1481*R<I-2>;

REM************************************************
   * GAIN AND LIMIT CALCULATIONS                  *
   ************************************************;

IF R<I> GT 16/.18 THEN R<I>=16/.18;
IF R<I> LT -16/.18 THEN R<I>=-16/.19;

REM************************************************
   * ACTUATOR DIFFERENCE EUATIONS                 *
   ************************************************;

A<I>=R<I>-F<I>-.667*(R<I-1>-F<I-1>);
A<I>=.2184*A<I>;

REM************************************************
   * ACTUATOR GAIN AND LIMIT CALCULATIONS         *
   ************************************************;

IF A<I> GT 22.5/.18 THEN A<I>=22.5/.18;
IF A<I> LT -22.5/.18 THEN A<I>=-22.5/.18;

REM************************************************
   *  FIN SERVO CALCULATIONS                      *
   ************************************************

F<I>=.6*A<I>-.4*F<I>;

REM************************************************
   * MISSILE FLIGHT CALCULATIONS                  *
   *                                              *
   * NATURAL FREQUENCY= 120 RADIANS/SEC.          *
   *                                              *
   *                                              *
   * DAMPING FACTOR=.7                            *
   *                                              *
   * GAIN=.2                                      *
   ************************************************;

L=1.24882;
E=0.313568;
F=0.458459;
G=0.30851;

C<I>=.2*K*E*F<I>+2*L*C<I-1>-G*C<I-2>;
X<I>=.2*C<I>+X<I>;
C<I>=S<I>-X<I>;
```

```
REM*************************************************
   *  TUMBLE VARIABLES                              *
   *************************************************;

R<1>=R<2>;
R<2>=R<3>;
C<1>=C<2>;
C<2>=C<3>;
A<1>=A<2>;
A<2>=A<3>;
F<1>=F<2>;
F<2>=F<3>;

REM*************************************************
   *          INPUT/OUTPUT SEQUENCES               *
   *                                               *
   *    1) INPUT COMMAND  (TESTER DRIVES BUS)      *
   *    2) FIN   COMMAND  (SUT    DRIVES BUS)      *
   *    3) FIN FEEDBACK   (TESTER DRIVES BUS)      *
   *    4) GYRO FEEDBACK  (TESTER DRIVES BUS)      *
   *************************************************;

REM INPUT 1'ST INPUT PATTERN;

DUT<DATA>=S<I>/.18 AND 377B; REM RESCALES AND COVERTS
DUT<DADB>=0;
DUT<MAMB>=1;  REM TESTER INPUTTING DATA ON THE BUS
                                  TO AN INTEGER NUMBER;

EXEC LMLRN(4,M,DUT);    REM GENERATES A TEST;
M=M+1;

REM OUTPUT ACTUATOR COMMAND;

DUT<DATA>=A<I>/.18 AND 377B;
DUT<DADB>=1;
DUT<MAMB>=0;    REM TESTER LOOKING AT DATA ON BUS;
EXEC LMLRN(4,M,DUT);
M=M+1;

REM INPUT FEEDBACK FROM FIN SERVO;

DUT<DATA>=F<I>/.18 AND 377B;
DUT<DADB>=0;      REM TESTER DRIVES THE BUS;
DUT<MAMB>=1;
EXEC LMLRN(4,M,DUT);
M=M+1;

REM INPUT GYRO FEEDBACK;

DUT<DATA>=C<I>/.18 AND 377B;
EXEC LMLRN(4,M,DUT);
M=M+1;
     END;                REM END OF FOR LOOP;
```

```
END;                                    REM END OF TEST PROGRAM;
```

```
****        FAIRCHILD FACTOR COMPILER          REL:   3.1          ****
**  SOURCE FILE:    *ED                           DATE:               **
**  DATA   FILE:                                  TIME:     00:35     **
**  SYSTEM CONF:    88      MV1     PMU4          COND:   00000000    **

 1                  SET PAGE 4095;
 2
 2                  REM ***********************************************
 2                      * SET UP PIN DEFINITIONS FOR ALGORITHMIC      *
 2                      * TEST PATTERN GENERATION                     *
 2                      ***********************************************;
 2
 2                  DCL PDATA[8]/10,11,12,13,14,15,16,17/;
 3                  DCL PHLD[1]/5/;
 4                  DCL PRDY[1]/3/;
 5
 5                  DCL DUT[5];
 6
 6                   DATA=1;
 7                   HLD=2;
 8                   RDY=3;
 9                   DADB=4;
10                   MAMB=5;
11
11                  EXEC LMMOD(0);    REM INITIALIZE TESTER MEMORY;
12
12                  EXEC LMLRN(1,PDATA,PHLD,PRDY,1,2);
13
13                  REM SET UP PIN DEFINITIONS IN TESTER;
13
13                  REM***********************************************
13                     *   DEFINE SIMULATION VARIABLES               *
13                     ***********************************************;
13
13                  DCL R[3];
14                  DCL C[3];
15                  DCL A[3];
16                  DCL F[3];
17                  DCL S[3];
18                  DCL X[3];
19
19                  REM***********************************************
19                     *  SIMULATION OF CONTROLLER USING DIFF. EQU.   *
19                     ***********************************************;
19
19                  I=3;
20                  DUT[HLD]=0;          REM START PROCESSOR;
21                  DUT[RDY]=1;
22                  DUT[DADB]=0;
23                  DUT[MAMB]=1;
24
24                  EXEC LMLRN(4,M,DUT);     REM GENERATE A TEST;
25
25                  M=M+1;                   REM TEST NUMBER COUNTER;
26
26                  FOR J=1 THRU 25 DO BEGIN
27          IF J EQ 3 THEN S[I]=10;
29
29                  R[I]=C[I]+.0818*C[I-1]-.9182*C[I-2]-.4132*R[I-1]
30                       +.1481*R[I-2];
30
30                  REM***********************************************
```

Figure 12. Compile Source Code

```
30                    * GAIN AND LIMIT CALCULATIONS                *



30                    ************************************************;
30
30                  IF R[I] GT 16/.18 THEN R[I]=16/.18;
32                  IF R[I] LT -16/.18 THEN R[I]=-16/.19;
34
34                  REM************************************************
34                     * ACTUATOR DIFFERENCE EUATIONS                 *
34                     ************************************************;
34
34                  A[I]=R[I]-F[I]-.667*(R[I-1]-F[I-1]);
35                  A[I]=.2184*A[I];
36
36                  REM************************************************
36            C        * ACTUATOR GAIN AND LIMIT CALCULATIONS        *
36                     ************************************************;
36
36                  IF A[I] GT 22.5/.18 THEN A[I]=22.5/.18;
38                  IF A[I] LT -22.5/.18 THEN A[I]=-22.5/.18;
40
40                  REM************************************************
40                     *  FIN SERVO CALCULATIONS                      *
40                     ************************************************;
40
40                  F[I]=.6*A[I]-.4*F[I];
41
41                  REM************************************************
41                     * MISSILE FLIGHT CALCULATIONS                  *
41                     *                                              *
41                     * NATURAL FREQUENCY= 120 RADIANS/SEC.          *
41                     *                                              *
41                     *                                              *
41                     * DAMPING FACTOR=.7                            *
41                     *                                              *
41                     * GAIN=.2                                      *
41                     ************************************************;
41
41                  K=1.24882;
42                  E=0.313568;
43                  L=0.458459;
44                  G=0.30851;
45
45                  C[I]=.2*K*E*F[I]+2*L*C[I-1]-G*C[I-2];
46                  X[I]=.2*C[I]+X[I];
47                  C[I]=S[I]-X[I];
48
48                  REM************************************************
48                     *  TUMBLE VARIABLES                            *
48                     ************************************************;
48
48                  R[1]=R[2];
49                  R[2]=R[3];
50                  C[1]=C[2];
51                  C[2]=C[3];
52                  A[1]=A[2];
53                  A[2]=A[3];
54                  F[1]=F[2];
55                  F[2]=F[3];
56
56          WRITE(POD)R[3],C[3],A[3],F[3],A[3]/.18 AND 377B,(A[3]/.18) AND 3
77B;
57                  REM************************************************
57                     *          INPUT/OUTPUT SEQUENCES              *
```

Figure 12. Compile Source Code (Continued)

```
 57                    *                                                *


 57                    *    1) INPUT COMMAND  (TESTER DRIVES BUS)       *
 57                    *    2) FIN   COMMAND  (SUT    DRIVES BUS)       *
 57                    *    3) FIN FEEDBACK   (TESTER DRIVES BUS)       *
 57                    *    4) GYRO FEEDBACK  (TESTER DRIVES BUS)       *
 57                    **************************************************;
 57
 57                  REM INPUT 1'ST INPUT PATTERN;
 57
 57                  DUT[DATA]=S[I]/.18 AND 377B; REM RESCALES AND COVERTS
 58                  DUT[DADB]=0;
 58                  DUT[MAMB]=1;  REM TESTER INPUTTING DATA ON THE BUS
 59                                                TO AN INTEGER NUMBER;
 59
 59                  EXEC LMLRN(4,M,DUT);     REM GENERATES A TEST;
 60                  M=M+1;
 61
 61                  REM OUTPUT ACTUATOR COMMAND;
 61
 61                  DUT[DATA]=A[I]/.18 AND 377B;
 62                  DUT[DADB]=1;
 63                  DUT[MAMB]=0;     REM TESTER LOOKING AT DATA ON BUS;
 64                  EXEC LMLRN(4,M,DUT);
 65                  M=M+1;
 66
 66                  REM INPUT FEEDBACK FROM FIN SERVO;
 66
 66                  DUT[DATA]=F[I]/.18 AND 377B;
 67                  DUT[DADB]=0;       REM TESTER DRIVES THE BUS;
 68                  DUT[MAMB]=1;
 69                  EXEC LMLRN(4,M,DUT);
 70                  M=M+1;
 71
 71                  REM INPUT GYRO FEEDBACK;
 71
 71                  DUT[DATA]=C[I]/.18 AND 377B;
 72                  EXEC LMLRN(4,M,DUT);
 73                  M=M+1;
 74                       END;                  REM END OF FOR LOOP;
 74
 74                  END;                       REM END OF TEST PROGRAM;
0  COMPILATION ERRORS,        74  STATEMENTS
```

Figure 12. Compile Source Code (Concluded)

```
 0 SET F     DA MB 0010000000 0000000000 0000000000
 1 SET F     DA MB 0010000000 0000000000 0000000000
 2 SET F     DB MA 0010000000 0000000000 0000000000
 3 SET F     DA MB 0010000000 0000000000 0000000000
 4 SET F     DA MB 0010000000 0000000000 0000000000
 5 SET F     DA MB 0010000000 0000000000 0000000000
 6 SET F     DB MA 0010000000 0000000000 0000000000
 7 SET F     DA MB 0010000000 0000000000 0000000000
 8 SET F     DA MB 0010000000 0000000000 0000000000
 9 SET F     DA MB 0010000000 0011100000 0000000000
10 SET F     DB MA 0010000000 0000000000 0000000000
11 SET F     DA MB 0010000000 0000000000 0000000000
12 SET F     DA MB 0010000000 0011100000 0000000000
13 SET F     DA MB 0010000000 0011100000 0000000000
14 SET F     DB MA 0010000001 0110000000 0000000000
15 SET F     DA MB 0010000000 0010000000 0000000000
16 SET F     DA MB 0010000001 0110100000 0000000000
17 SET F     DA MB 0010000000 0011100000 0000000000
18 SET F     DB MA 0010000001 0001111000 0000000000
19 SET F     DA MB 0010000000 0101111000 0000000000
20 SET F     DA MB 0010000001 0010100000 0000000000
21 SET F     DA MB 0010000000 0011100000 0000000000
22 SET F     DB MA 0010000000 1010000000 0000000000
23 SET F     DA MB 0010000001 1010000000 0000000000
24 SET F     DA MB 0010000000 0100100000 0000000000
25 SET F     DA MB 0010000000 0011100000 0000000000
26 SET F     DB MA 0010000001 0011111000 0000000000
27 SET F     DA MB 0010000000 0011111000 0000000000
28 SET F     DA MB 0010000000 0000100000 0000000000
29 SET F     DA MB 0010000000 0011100000 0000000000
30 SET F     DB MA 0010000000 0100000000 0000000000
31 SET F     DA MB 0010000000 1100000000 0000000000
32 SET F     DA MB 0010000000 0111000000 0000000000
33 SET F     DA MB 0010000000 0011100000 0000000000
34 SET F     DB MA 0010000001 0111111000 0000000000
35 SET F     DA MB 0010000000 0111111000 0000000000
36 SET F     DA MB 0010000001 0011000000 0000000000
37 SET F     DA MB 0010000000 0011100000 0000000000
38 SET F     DB MA 0010000000 1000000000 0000000000
39 SET F     DA MB 0010000001 1000000000 0000000000
40 SET F     DA MB 0010000000 1101000000 0000000000
41 SET F     DA MB 0010000000 0011100000 0000000000
42 SET F     DB MA 0010000001 1111111000 0000000000
43 SET F     DA MB 0010000000 1111111000 0000000000
44 SET F     DA MB 0010000000 0101000000 0000000000
45 SET F     DA MB 0010000000 0011100000 0000000000
46 SET F     DB MA 0010000001 0000000000 0000000000
47 SET F     DA MB 0010000001 0000000000 0000000000
48 SET F     DA MB 0010000001 0001000000 0000000000
49 SET F     DA MB 0010000000 0011100000 0000000000
50 SET F     DB MA 0010000001 1111111000 0000000000
51 SET F     DA MB 0010000001 1111111000 0000000000
52 SET F     DA MB 0010000001 1110000000 0000000000
53 SET F     DA MB 0010000000 0011100000 0000000000
54 SET F     DB MA 0010000000 0000000000 0000000000
55 SET F     DA MB 0010000001 0000000000 0000000000
56 SET F     DA MB 0010000000 1110000000 0000000000
57 SET F     DA MB 0010000000 0011100000 0000000000
58 SET F     DB MA 0010000000 0000000000 0000000000
59 SET F     DA MB 0010000000 0000000000 0000000000
60 SET F     DA MB 0010000000 0110000000 0000000000
61 SET F     DA MB 0010000000 0011100000 0000000000
```

Figure 13. Resulting Test Sequence